W. Kluge

Abstract Computing Machines

A Lambda Calculus Perspective

With 89 Figures

Springer

Authors

Prof. Dr. Werner Kluge
Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel
Olshausenstrasse 40b, 24098 Kiel, Germany
wk@informatik.uni-kiel.de

Library of Congress Control Number: 2004117887

ACM Computing Classification (1998): D.3.2, D.3.4, F.3

ISBN 3-540-21146-2 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media
springeronline.com

Cover design: KünkelLopka, Heidelberg
Typesetting: Camera ready by authors
Production: LE-TeX Jelonek, Schmidt & Vöckler GbR, Leipzig
Printed on acid-free paper SPIN: 10991046 45/3142/YL - 5 4 3 2 1 0

Klaus Berkling

1931 – 1997

Preface

This monograph looks at computer organization from a strictly conceptual point of view to identify the very basic mechanisms and runtime structures necessary to perform algorithmically specified computations. It completely abstracts from concrete programming languages and machine architectures, taking the λ-calculus – a theory of computable functions – as the basic programming and program execution model. In its simplest form, the λ-calculus talks about expressions that are constructed from just three syntactical figures – variables, functions (in this context called abstractions) and applications (of operator to operand expressions) – and about a single transformation rule that governs the substition of variable occurrences in expressions by other expressions. This β-reduction rule contains in a nutshell the whole story about computing, specifically about the role of variables and variable scoping in this game.

Different implementations of the β-reduction rule in conjunction with strategies that define the sequencing of β-reductions in complex expressions give rise to a variety of abstract λ-calculus machines that are studied in this text. These machines share, in one way or another, the components of Landin's SECD machine – a program text to be executed, a runtime environment that holds delayed substitutions, a value stack, and a dump stack for return continuations – but differ with respect to the internal representation of λ-expressions, specifically abstractions, the structure of the runtime environments and the mechanisms of program execution.

This text covers more than just implementations of functional or function-based languages such as MIRANDA, HASKELL, CLEAN, ML or SCHEME which realize what is called a weakly normalizing λ-calculus that uses a naive version of the β-reduction rule. The emphasis is instead on λ-calculus machines that are fully normalizing, using a complete and correct implementation of the β-reduction rule, which includes the orderly resolution of naming conflicts that may occur when free variables are substituted under abstractions. This feature is an essential prerequisite for correct symbolic computations that treat both functions and variables truly as first-class objects. It may, for instance, be

used to advantage in theorem provers to establish equality between two terms that contain variables, or to symbolically simplify expressions in the process of high-level program optimizations.

In weakly normalizing machines, the flavors of a full-fledged β-reduction are traded in for naive substitutions that are simpler to implement and require less complex runtime structures, resulting in improved runtime efficiency. Naming conflicts are consequently avoided by outlawing substitutions under abstractions, with the consequence that only ground terms (or basic values) can be computed. Weakly normalizing machines are therefore the standard vehicles for the implementation of functional or function-based languages whose semantics conform to this restriction. However, they are also used as integral parts of fully normalizing machines to perform the majority of those β-reductions that in fact can be carried out naively. Whenever substitutions need to be pushed under abstractions, a special mechanism equivalent to full β-reductions takes over to perform renaming operations that resolve potential name clashes.

Abstract machines for classical imperative languages are shown to be descendants of weakly normalizing machines that allow side-effecting operations, specified as assignments to bound variables, on the runtime emvironment. These side effects destroy important invariance properties of the λ-calculus that guarantee the determinacy of results irrespective of execution orders, leaving just the static scoping rules for bound variables intact. In this degenerate form of the λ-calculus, programs are primarily executed for their effects on the environment, as opposed to computing the values of the expressions of a weakly or fully normalizing λ-calculus.

This monograph, though not exactly mainstream, may be used in a graduate course on computer organization/architecture that focuses on the essentials of performing computations mechanically. It includes an introduction to the λ-calculus, specifically a nameless version suitable for machine implementation, and then continues to describe various fully and weakly normalizing λ-calculus machines at different levels of abstractions (direct interpretation, graph interpretation, execution of compiled code), followed by two kinds of abstract machines for imperative languages. The workings of these machines are specified by sets of state transition rules. The book also specifies, for code-executing abstract machines, compilation schemes that transform an applied λ-calculus taken as a reference source language to abstract machine code. Whenever deemed helpful, the execution of small example programs is also illustrated in a step-by-step fashion by sequences of machine state transitions.

I have used most of the material of this monograph in several graduate courses on computer organization which I taught over the years at the University of Kiel. Some of the material (Chaps. 2, 3 and the easier parts of Chaps. 4, 5) I even used in an undergraduate course on programming. The general impression was that at least the brighter students, after some time of getting used to the approach and to the notation, caught on pretty well to the message that I wanted to get across: understanding basic concepts and

principles of performing computations by machinery (with substitution as the most important operation) that are invariant against trendy ways of doing things in real computing machines, and how they relate to basic programming paradigms.

Acknowledgments

There are several people who contributed to this text with discussions and suggestions relating to its contents, with critical comments on earlier drafts, and with careful proofreading that uncovered many errors (of which some would have been somewhat embarrassing).

I am particularly indebted to Claus Reinke who gave Chaps. 5 to 8 and Appendix A a very thorough going-over, made some valuable recommendations that helped to improve verbal explanations and also the formal apparatus, specifically in Appendix A which I have largely adopted from his excellent PhD thesis, and provided me with a long list of ambiguities, notational inconsistencies and errors. Some intensive discussions with Sven-Bodo Scholz on head-order reduction, specifically on the problem of shared evaluation, led to substantial improvements of Chaps. 6 to 8. He also pointed out quite a few things in Chaps. 12 and 13 that needed clarification. I also had two enlightening discussions with Henk Barendregt and Rinus Plasmeijer on λ-calculus and on theorem proving which helped to shape Chaps. 4, 11 and Appendix B. Ulrich Bruening checked and made some helpful comments on Chaps. 13 and 14. Hans Langmaack was always available for some insightful discussions of language issues.

Makoto Amamiya gave me the opportunity to teach parts of this text in a one-week seminar course at Kyushu University in Fukuoka/Japan. The ensuing discussions gave me a fairly good idea of how the material would sink in with graduate students who have a slightly different background, and they also helped to correct a few flaws.

Kay Berkling, Claudia Schmittgen and Erich Valkema carefully proofread parts of a text that was more or less unfamiliar scientific territory to them, pointing out a few things that needed to be clarified, explained in more detail (by more examples), or simply corrected.

Last, not least, I wish to thank the people at Springer for their support of this project, especially Ingeborg Mayer, Ronan Nugent, Frank Holzwarth and, most importantly, Douglas Meekison who as a copyeditor did an excellent job of polishing the style of presentation, the layout of the text, and the English. There was hardly anything that escaped his attention.

... and there was Moni whose occasional peptalks kept me going.

Werner Kluge, November 2004

Contents